# Natural Language Processing

Last update: 11 October 2024

Academic Year 2024 – 2025

Alma Mater Studiorum · University of Bologna

# Contents

# 1 Basic text processing

**Text normalization** Operations such as:

**Tokenization** Split a sentence in tokens. Tokenization

| **Remark.** Depending on the approach, a token is not always a word.

**Lemmatization/stemming** Convert words to their canonical form. Lemmatization/stemming

| **Example.** $\{\texttt{sang}, \texttt{sung}, \texttt{sings}\} \mapsto \texttt{sing}$

**Sentence segmentation** Split a text in sentences. Sentence segmentation

| **Remark.** A period does not always signal the end of a sentence.

## 1.1 Regular expressions

**Regular expression (regex)** Formal language to describe string patterns. Regular expression (regex)

### 1.1.1 Basic operators

**Disjunction (brackets)** Match a single character between square brackets `[]`.

| **Example.** `/[wW]oodchuck/` matches `Woodchuck` and `woodchuck`.

**Range** Match a single character from a range of characters or digits.

| **Example.**
- `/[A-Z]/` matches a single upper case letter.
- `/[a-z]/` matches a single lower case letter.
- `/[0-9]/` matches a single digit.

**Negation** Match the negation of a pattern.

| **Example.** `/[^A-Z]/` matches a single character that is not an upper case letter.

**Disjunction (pipe)** Disjunction of regular expressions separated by `|`.

| **Example.** `/groundhog|woodchuck/` matches `groundhog` and `woodchuck`.

**Wildcards**

**Optional** A character followed by `?` can be matched optionally.

| **Example.** `/woodchucks?/` matches `woodchuck` and `woodchucks`.

**Any** `.` matches any character.

**Kleene** `*` A character followed by `*` can be matched zero or more times.

**Kleene** `+` A character followed by `+` must be matched at least once.

**Counting** A character followed by `{n,m}` must be matched from $n$ to $m$ times.

**Example.**

- `{n}` matches exactly $n$ instances of the previous character.
- `{n,m}` matches from $n$ to $m$ instances of the previous character.
- `{n,}` matches at least $n$ instances of the previous character.
- `{,m}` matches at most $m$ instances of the previous character.

**Anchors**

**Start of line** `^` matches only at the start of line.

**Example.** `/^a/` matches a but not `ba`.

**End of line** `$` matches only at the end of line.

**Example.** `/a$/` matches a but not `ab`.

**Word boundary** `\b` matches a word boundary character.

**Word non-boundary** `\B` matches a word non-boundary character.

**Aliases**

- `\d` matches a single digit (same as `[0-9]`).
- `\D` matches a single non-digit (same as `[^\d]`).
- `\w` matches a single alphanumeric or underscore character (same as `[a-zA-Z0-9_]`).
- `\w` matches a single non-alphanumeric and non-underscore character (same as `[^\w]`).
- `\s` matches a single whitespace (space or tab).
- `\S` matches a single non-whitespace.

**Capture group** Operator to refer to previously matched substrings.

**Example.** In the regex `/the (.*)er they were, the \1er they will be/`, `\1` should match the same content matched by `(.*)`.

## 1.2 Tokenization

**Lemma** Words with the same stem and roughly the same semantic meaning. Lemma

**Example.** `cat` and `cats` are the same lemma.

**Wordform** Orthographic appearance of a word. Wordform

**Example.** `cat` and `cats` do not have the same wordform.

**Vocabulary** Collection of text elements, each indexed by an integer. Vocabulary

**Remark.** To reduce the size of a vocabulary, words can be reduced to lemmas.

**Type / Wordtype** Element of a vocabulary (i.e., wordforms in the vocabulary). Type / Wordtype

**Token** Instance of a type in a text. Token

**Genre** Topic of a text corpus (e.g., short social media comments, books, Wikipedia pages, ...). Genre

**Remark** (Herdan's law)**.** Given a corpus with $N$ tokens, a vocabulary $V$ over that corpus roughly have size:

$$|V| = kN^{\beta}$$

where the typical values are $10 \leq k \leq 100$ and $0.4 \leq \beta \leq 0.6$.

**Stopwords** Frequent words that can be dropped. Stopwords

> **Remark.** If semantics is important, stopwords should be kept. LLMs keep stopwords.

**Rule-based tokenization** Hand-defined rules for tokenization. Rule-based tokenization

> **Remark.** For speed, simple tokenizers use regex.

**Data-driven tokenization** Determine frequent tokens from a large text corpus. Data-driven tokenization

### 1.2.1 Data-driven tokenization

Tokenization is done by two components:

**Token learner** Learns a vocabulary from a given corpus (i.e., training). Token learner

**Token segmenter** Segments a given input into tokens based on a vocabulary (i.e., inference). Token segmenter

**Byte-pair encoding (BPE)** Based on the most frequent $n$-grams. Byte-pair encoding (BPE)

**Token learner** Given a training corpus $C$, BPE determines the vocabulary as follows:

1. Start with a vocabulary $V$ containing all the 1-grams of $C$ and an empty set of merge rules $M$.
2. While the desired size of the vocabulary has not been reached:
   a) Determine the pair of tokens $t_1 \in V$ and $t_2 \in V$ such that, among all the possible pairs, the $n$-gram $t_1 + t_2 = t_1 t_2$ obtained by merging them is the most frequent in the corpus $C$.
   b) Add $t_1 t_2$ to $V$ and the merge rule $t_1 + t_2$ to $M$.

> **Example.** Given the following corpus:
>
> | Occurrences | Tokens |
> |---|---|
> | 5 | `l o w $` |
> | 2 | `l o w e r $` |
> | 6 | `n e w e s t $` |
> | 6 | `w i d e s t $` |
>
> The initial vocabulary is: $V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\}$.
> At the first iteration, $\texttt{e} + \texttt{s} = \texttt{es}$ is the most frequent $n$-gram. Corpus and vocabulary are updated as:

| Occurrences | Tokens |
|---|---|
| 5 | `l o w $` |
| 2 | `l o w e r $` |
| 6 | `n e w es t $` |
| 6 | `w i d es t $` |

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\} \cup \{\texttt{es}\}$$

At the second iteration, $\texttt{es} + \texttt{t} = \texttt{est}$ is the most frequent $n$-gram:

| Occurrences | Tokens |
|---|---|
| 5 | `l o w $` |
| 2 | `l o w e r $` |
| 6 | `n e w est $` |
| 6 | `w i d est $` |

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}, \texttt{es}\} \cup \{\texttt{est}\}$$

And so on...

**Token segmenter** Given the vocabulary $V$ and the merge rules $M$, the BPE segmenter does the following:

1. Split the input into 1-grams.
2. Iteratively scan the input and do the following:
   a) Apply a merge rule if possible.
   b) If no merge rules can be applied, lookup the (sub)word in the vocabulary. Tokens out-of-vocabulary are marked with a special unknown token `[UNK]`.

**WordPiece** Similar to BPE with the addition of merge rules ranking and a special leading/tailing set of characters (usually `##`) to identify subwords (e.g., `new##`, `##est` are possible tokens). WordPiece

**Unigram** Starts with a big vocabulary and remove tokens following a loss function. Unigram

## 1.3 Normalization

**Normalization** Convert tokens into a standard form. Normalization

**Example.** `U.S.A.` and `USA` should be encoded using the same index.

**Case folding** Map every token to upper/lower case. Case folding

**Remark.** Depending on the task, casing might be important (e.g., `US` vs `us`).

**Lemmatization** Reduce inflections and variant forms to their base form. Lemmatization

**Example.** $\{\texttt{am}, \texttt{are}, \texttt{is}\} \mapsto \texttt{be}$

**Remark.** Accurate lemmatization requires complete morphological parsing.

**Stemming** Reduce terms to their stem. Stemming

**Remark.** Stemming is a simpler approach to lemmatization.

**Porter stemmer** Simple stemmer based on cascading rewrite rules.

> **Example.** `ational` $\mapsto$ `ate`, `ing` $\mapsto \varepsilon$, `sses` $\mapsto$ `ss`.

## 1.4 Edit distance

**Minimum edit distance** Minimum number of edit operations (insertions, deletions, and substitutions) needed to transform a string into another one.

Minimum edit distance

> **Remark.** Dynamic programming can be used to efficiently determine the minimum edit distance.

**Levenshtein distance** Edit distance where:

Levenshtein distance

- Insertions cost 1;
- Deletions cost 1;
- Substitutions cost 2.

> **Example.** The Levenshtein distance between `intention` and `execution` is 8.

```
I  N  T  E  *  N  T  I  O  N
|  |  |  |  |  |  |  |  |  |
*  E  X  E  C  U  T  I  O  N
−  ±  ±     +  ±
1  2  2     1  2
```

# 2 Language models

## 2.1 Spelling correction

Spelling correction

**Spelling correction** Spelling errors can be of two types:

**Non-word spelling** Typos that result in non-existing words. Possible candidates can be determined though a dictionary lookup.

**Real-word spelling** Can be:

**Typographical error** Typos that result in existing words.

**Cognitive error** Due to words similarity (e.g., `piece` vs `peace`).

Noisy channel model

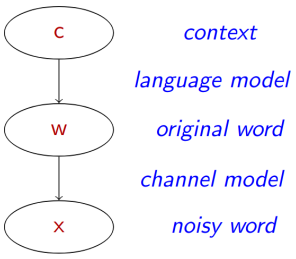
**Noisy channel model** Assumes that the observable input is a distorted form of the original word. A decoder tests word hypotheses and selects the best match.

More formally, we want a model of the channel that, similarly to Bayesian inference, determines the likelihood that a word $w \in V$ is the original word for a noisy one $x$. From there, we can estimate the correct word $\hat{w}$:

$$\hat{w} = \arg\max_{w \in V} \mathcal{P}(w|x)$$


By applying (i) Bayes' rule, (ii) the fact that $\hat{w}$ is independent of $\mathcal{P}(x)$, and (iii) that a subset $C \subseteq V$ of the vocabulary can be used, the estimate becomes:

$$\hat{w} = \arg\max_{w \in C} \underbrace{\mathcal{P}(x|w)}_{\text{channel model}} \underbrace{\mathcal{P}(w)}_{\text{prior}}$$




Moreover, it is reasonable to include a context $c$ when computing the prior:

$$\hat{w} = \arg\max_{w \in C} \underbrace{\mathcal{P}(x|w)}_{\text{channel model}} \underbrace{\mathcal{P}(w|c)}_{\text{language model}}$$

**Noisy channel spelling method** Spelling correction in a noisy channel model can be done as follows:

1. Find candidate words with similar spelling to the input based on a distance metric (e.g., Damerau-Levenshtein which is the Levenshtein distance with the addition of adjacent transpositions).
2. Score each candidate based on the language and channel model:
   - Use typing features of the user.
   - Use local context.
   - Use a confusion matrix with common mistakes.

**Example.** Consider the sentence:

> [...] *was called a "stellar and versatile* `acress` *whose combination of sass and glamour has defined her* [...]*"*

By using the Corpus of Contemporary English (COCA), we can determine the following words as candidates:

$$\texttt{actress} \cdot \texttt{cress} \cdot \texttt{caress} \cdot \texttt{access} \cdot \texttt{across} \cdot \texttt{acres}$$

**Language model** By considering a language model without context, the priors are computed as $\mathcal{P}(w) = \frac{\texttt{count}(w)}{|\texttt{COCA}|}$ (where $|\texttt{COCA}| = 404\,253\,213$):

| $w$ | $\texttt{count}(w)$ | $\mathcal{P}(w)$ |
|---|---|---|
| actress | 9321 | 0.0000231 |
| cress | 220 | 0.000000544 |
| caress | 686 | 0.00000170 |
| access | 37 038 | 0.0000916 |
| across | 120 844 | 0.000299 |
| acres | 12 874 | 0.0000318 |

**Channel model** By using a confusion matrix of common typos, the channel model is:

| $w$ | $x\|w$ | $\mathcal{P}(x\|w)$ |
|---|---|---|
| actress | c\|ct | 0.000117 |
| cress | a\|# | 0.00000144 |
| caress | ac\|ca | 0.00000164 |
| access | r\|c | 0.000000209 |
| across | e\|o | 0.0000093 |
| acres | es\|e | 0.0000321 |
| acres | ss\|s | 0.0000342 |

The ranking is the obtained as:

| $w$ | $\mathcal{P}(x\|w)\mathcal{P}(w)$ |
|---|---|
| actress | $2.7 \cdot 10^9$ |
| cress | $0.00078 \cdot 10^9$ |
| caress | $0.0028 \cdot 10^9$ |
| access | $0.019 \cdot 10^9$ |
| across | $2.8 \cdot 10^9$ |
| acres | $1.02 \cdot 10^9$ |
| acres | $1.09 \cdot 10^9$ |

Therefore, the most likely correction of `acress` for this model is `across`.
If the previous word is considered in the context, the relevant tokens of the new language model are:

| $w_{i-1}$ | $w_i$ | $\mathcal{P}(w_i\|w_{i-1})$ |
|---|---|---|
| versatile | actress | 0.000021 |
| versatile | across | 0.000021 |
| actress | whose | 0.001 |
| across | whose | 0.000006 |

This allows to measure the likelihood of a sentence as:

$$\mathcal{P}(\texttt{versatile actress whose}) = \mathcal{P}(\texttt{actress}|\texttt{versatile})\mathcal{P}(\texttt{whose}|\texttt{actress}) = 210 \cdot 10^{-10}$$
$$\mathcal{P}(\texttt{versatile across whose}) = \mathcal{P}(\texttt{across}|\texttt{versatile})\mathcal{P}(\texttt{whose}|\texttt{across}) = 1 \cdot 10^{-10}$$

Finally, we have that:

$$\mathcal{P}(\texttt{versatile actress whose}|\texttt{versatile acress whose}) = 2.7 \cdot 210 \cdot 10^{-19}$$
$$\mathcal{P}(\texttt{versatile across whose}|\texttt{versatile acress whose}) = 2.8 \cdot 10^{-19}$$

So `actress` is the most likely correction for `acress` in this model.

**Remark.** In practice, log-probabilities are used to avoid underflows and to make computation faster (i.e., sums instead of products).

## 2.2 Language models

**(Probabilistic) language model** Model to determine the probability of a word $w$ in a given context $c$: Language model

$$\mathcal{P}(w|c)$$

Usually, it is based on counting statistics and uses as context the sequence of previous tokens:

$$\mathcal{P}(w_i|w_1, \dots, w_{i-1})$$

This is equivalent to computing the probability of the whole sentence, which expanded using the chain rule becomes:

$$\begin{aligned}\mathcal{P}(w_1, \dots, w_{i-1}w_i) &= \mathcal{P}(w_1)\mathcal{P}(w_2|w_1)\mathcal{P}(w_3|w_{1..2}) \dots \mathcal{P}(w_n|w_{1..n-1}) \\ &= \prod_{i=1}^{n} \mathcal{P}(w_i|w_{1..i-1})\end{aligned}$$

**Remark.** Simply counting the number of occurrences of a sentence as $\mathcal{P}(w_i|w_{1..i-1}) = w_{1..i}/w_{1..i-1}$ is not ideal as there are too many possible sentences.

**Markov assumption** Limit the length of the context to a window of $k$ previous tokens: Markov assumption in language models

$$\mathcal{P}(w_i|w_{1..i-1}) \approx \mathcal{P}(w_i|w_{i-k..i-1})$$
$$\mathcal{P}(w_{1..n}) \approx \prod_{i=1}^{n} \mathcal{P}(w_i|w_{i-k..i-1})$$

**Unigram model** Model without context ($k = 0$):

$$\mathcal{P}(w_{1..n}) \approx \prod_i \mathcal{P}(w_i)$$

**Bigram model** Model with a single token context ($k = 1$):

$$\mathcal{P}(w_{1..n}) \approx \prod_i \mathcal{P}(w_i|w_{i-1})$$

**N-gram model** Model with a context of $k = N - 1$ tokens: $N$-gram model

$$\mathcal{P}(w_{1..n}) \approx \prod_i \mathcal{P}(w_i|w_{i-N+1..i-1})$$

**Remark.** $N$-gram models cannot capture long-range dependencies.

**Estimating N-gram probabilities** Consider the bigram case, the probability that a token $w_i$ follows $w_{i-1}$ can be determined by counting:

$$\mathcal{P}(w_i|w_{i-1}) = \frac{\texttt{count}(w_{i-1}w_i)}{\texttt{count}(w_{i-1})}$$

**Remark.** $N$-gram models cannot handle unknown tokens.

**Remark.** $N$-gram models capture knowledge about:

- Grammar and syntax.
- Some information about the dataset (e.g., domain, genre of corpus, cultural aspects, ...).

**Generation by sampling** Randomly sample tokens from the distribution of a language model. Generation by sampling

**Remark.** In $N$-gram models ($N \geq 2$), the distribution changes depending on the previously sampled tokens.

## 2.3 Metrics

### 2.3.1 Extrinsic evaluation

**Extrinsic/downstream evaluation** Compare the performance of different models on specific tasks. Extrinsic evaluation

**Remark.** Extrinsic evaluation is the best approach for comparing different models, but it is often computationally expensive.

### 2.3.2 Intrinsic evaluation

**Intrinsic evaluation** Measure the quality of a model independently of the task. Intrinsic evaluation

**Perplexity (PP)** Probability-based metric based on the inverse probability of a sequence (usually the test set) normalized by the number of words: Perplexity

$$\mathcal{P}(w_{1..N}) = \prod_i \mathcal{P}(w_i|w_{1..i-1})$$

$$\texttt{PP}(w_{1..N}) = \mathcal{P}(w_{1..N})^{-\frac{1}{N}} \in [1, +\infty]$$

A lower perplexity represents a generally better model.

**Example.** For bigram models, perplexity is computed as:

$$\mathcal{P}(w_{1..N}) \approx \prod_i \mathcal{P}(w_i|w_{i-1}) \qquad \texttt{PP}(w_{1..N}) = \sqrt[N]{\prod_i \frac{1}{\mathcal{P}(w_i|w_{i-1})}}$$

**Remark** (Perplexity intuition)**.** Perplexity can be seen as a measure of surprise of a language model when evaluating a sequence.

Alternatively, it can also be seen as a weighted average branching factor (i.e., average number of possible next words that follow any word, accounting for their probabilities). For instance, consider a vocabulary of digits and a training corpus where every digit appears with uniform probability 0.1. The perplexity of any sequence using a 1-gram model is:

$$\texttt{PP}(w_{1..N}) = \left(0.1^N\right)^{-\frac{1}{N}} = 10$$

Now consider a training corpus where 0 occurs 91% of the time and the other digits 1% of the time. The perplexity of the sequence `0 0 0 0 0 3 0 0 0 0` is:

$$\texttt{PP}(\texttt{0 0 0 0 0 3 0 0 0 0}) = \left(0.91^9 \cdot 0.01\right)^{-\frac{1}{10}} \approx 1.73$$

**Remark.** Minimizing perplexity is the same as maximizing the probability of the tokens.

**Remark.** Perplexity can be artificially reduced by using a smaller vocabulary. Therefore, it is only reasonable to compare perplexity of models with the same vocabulary.

**Remark.** Perplexity is generally a bad approximation of extrinsic metrics and only works well if the test set is representative of the training data. Therefore, it is only useful to guide experiments and the final evaluation should be done through extrinsic evaluation.

## 2.4 N-gram model problems

### 2.4.1 Overfitting

Overfitting

$N$-gram models become better at modeling the training corpus for increasing values of $N$. This risks overfitting and does not allow to obtain a generalized model.

**Example.** A 4-gram model is able to nearly perfectly generate sentences from Shakespeare's works.

### 2.4.2 Out-of-vocabulary tokens

There are two types of vocabulary systems:

Closed vocabulary

**Closed vocabulary system** All words that can occur are known.

Open vocabulary

**Open vocabulary system** Unknown words are possible. They are usually handled using a dedicated token `<UNK>` which allows to turn an open vocabulary system into a closed one:

- Use a vocabulary and model all other words as `<UNK>`.
- Model infrequent words as `<UNK>`.

**Remark.** The training set must contain `<UNK>` tokens to estimate its distribution as it is treated as any other token.

### 2.4.3 Unseen sequences

Only for $n$-grams that occur enough times a representative probability can be estimated. For increasing values of $n$, the sparsity grows causing many unseen $n$-grams that produce a probability of 0, with the risk of performing divisions by zero (e.g., perplexity) or zeroing probabilities (e.g., when applying the chain rule).

**Laplace smoothing** Adds 1 to all counts and renormalizes them. Given a vocabulary $V$ and an $N$-gram model, smoothing is done as follows: Laplace smoothing

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-N+1..i-1}) = \frac{\texttt{count}(w_{i-N+1..i-1}w_i)+1}{\texttt{count}(w_{i-N+1..i-1})+|V|}$$

Alternatively, by only changing the numerator, it can be formulated using an adjusted count as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-N+1..i-1}) = \frac{c^*}{\texttt{count}(w_{i-N+1..i-1})}$$
$$c^* = \big(\texttt{count}(w_{i-N+1..i-1}w_i)+1\big)\frac{\texttt{count}(w_{i-N+1..i-1})}{\texttt{count}(w_{i-N+1..i-1})+|V|}$$

where $\frac{\texttt{count}(w_{i-N+1..i-1})}{\texttt{count}(w_{i-N+1..i-1})+|V|}$ is a normalization factor.

**Example.** For a 2-gram model, Laplace smoothing is computed as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-1}) = \frac{\texttt{count}(w_{i-1}w_i)+1}{\texttt{count}(w_{i-1})+|V|}$$

Or by using the adjusted count as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-1}) = \frac{c^*}{\texttt{count}(w_{i-1})} \qquad c^* = \big(\texttt{count}(w_{i-1}w_i)+1\big)\frac{\texttt{count}(w_{i-1})}{\texttt{count}(w_{i-1})+|V|}$$

# 3 Text classification

## 3.1 Common tasks

**Sentiment analysis/Opinion mining** Detection of attitudes. It can involve detecting: Sentiment analysis/Opinion mining

- The holder of attitude (i.e., the source).
- The target of attitude (i.e., the aspect).
- The type of attitude (e.g., positive and negative).
- The text containing the attitude.

**Spam detection**

**Language identification**

**Authorship attribution**

**Subject category classification**

## 3.2 Classification

**Classification task** Given an input $x$ and a set of possible classes $Y = \{y_1, \dots, y_M\}$, a classifier determines the class $\hat{y} \in Y$ associated to $x$. Classification task

Classification can be:

**Rule-based** Based on fixed (possibly handwritten) rules. Rule-based

| **Example.** Blacklist, whitelist, regex, ...

**In-context learning** Provide a decoder (i.e., generative) large language model a prompt describing the task and the possible classes. In-context learning

| **Example.** Zero-shot learning, few-shot learning, ...

**Supervised machine learning** Use a training set of $N$ labeled data $\{(d_i, c_i)\}$ to fit a classifier. Supervised machine learning

An ML model can be:

**Generative** Informally, it learns the distribution of the data.

**Discriminative** Informally, it learns to exploit the features to determine the class.

## 3.3 Naive Bayes

**Bag-of-words (BoW)** Represents a document using the frequencies of its words. Bag-of-words (BoW)

Given a vocabulary $V$ and a document $d$, the bag-of-words embedding of $d$ is a vector in $\mathbb{N}^{|V|}$ where the $i$-th position contains the number of occurrences of the $i$-th token in $d$.

**Multinomial naive Bayes classifier** Probabilistic (i.e., generative) classifier based on the assumption that features are independent given the class.

Given a document $d = \{w_1, \ldots, w_n\}$, a naive Bayes classifier returns the class $\hat{c}$ with maximum posterior probability:

$$\begin{aligned}
\hat{c} &= \arg\max_{c \in C} \mathcal{P}(c|d) \\
&= \arg\max_{c \in C} \underbrace{\mathcal{P}(d|c)}_{\text{likelihood}} \underbrace{\mathcal{P}(c)}_{\text{prior}} \\
&= \arg\max_{c \in C} \mathcal{P}(w_1, \ldots, w_n|c)\mathcal{P}(c) \\
&= \arg\max_{c \in C} \prod_i \mathcal{P}(w_i|c)\mathcal{P}(c) \\
&= \arg\max_{c \in C} \sum_i \log \mathcal{P}(w_i|c) \log \mathcal{P}(c)
\end{aligned}$$

Given a training set $D$ with $N_c$ classes and a vocabulary $V$, $\mathcal{P}(w_i|c)$ and $\mathcal{P}(c)$ are determined during training by maximum likelihood estimation as follows:

$$\mathcal{P}(c) = \frac{N_c}{|D|} \qquad \mathcal{P}(w_i|c) = \frac{\texttt{count}(w_i, c)}{\sum_{v \in V} \texttt{count}(v, c)}$$

where $\texttt{count}(w, c)$ counts the occurrences of the word $w$ in the training samples with class $c$.

**Remark.** Laplace smoothing is used to avoid zero probabilities.

**Remark.** Stop words can be removed from the training set as they are usually not relevant.

**Remark.** The likelihood part of the equation ($\sum_i \log \mathcal{P}(w_i|c)$) can be seen as a set of class-specific 1-gram language models.

**Example.** Given the following training set for sentiment analysis with two classes:

| Class | Document |
|---|---|
| - | `just plain boring` |
| - | `entirely predictable and lacks energy` |
| - | `no surprises and very few laughs` |
| + | `very powerful` |
| + | `the most fun film of the summer` |

We want to classify the sentence "`predictable with no fun`". Excluding stop words (i.e., `with`), we need to compute:

$$\mathcal{P}(\texttt{+}|\texttt{predictable with no fun}) = \mathcal{P}(\texttt{+})\mathcal{P}(\texttt{predictable}|\texttt{+})\mathcal{P}(\texttt{no}|\texttt{+})\mathcal{P}(\texttt{fun}|\texttt{+})$$
$$\mathcal{P}(\texttt{-}|\texttt{predictable with no fun}) = \mathcal{P}(\texttt{-})\mathcal{P}(\texttt{predictable}|\texttt{-})\mathcal{P}(\texttt{no}|\texttt{-})\mathcal{P}(\texttt{fun}|\texttt{-})$$

A vocabulary of 20 tokens can be used to represent the training samples. The required likelihoods and priors with Laplace smoothing are computed as:

$$\mathcal{P}(\texttt{+}) = \frac{2}{5} \qquad \mathcal{P}(\texttt{predictable}|\texttt{+}) = \frac{0+1}{9+20} \quad \mathcal{P}(\texttt{no}|\texttt{+}) = \frac{0+1}{9+20} \quad \mathcal{P}(\texttt{fun}|\texttt{+}) = \frac{1+1}{9+20}$$
$$\mathcal{P}(\texttt{-}) = \frac{3}{5} \qquad \mathcal{P}(\texttt{predictable}|\texttt{-}) = \frac{1+1}{14+20} \quad \mathcal{P}(\texttt{no}|\texttt{-}) = \frac{1+1}{14+20} \quad \mathcal{P}(\texttt{fun}|\texttt{-}) = \frac{0+1}{14+20}$$